An Electronic Design Assistance Tool for Case Based Representation of Designs

Ömer Akin, Michael Cumming, Michael Shealey, and Bige Tuncer
Department of Architecture, Carnegie Mellon University, Pittsburgh, PA

ABSTRACT

In precedent based design, solutions to problems are developed by drawing from an understanding of landmark designs. Many of the key design operations in this mode are similar to the functionalities present in case based reasoning systems: case matching, case adapting, and case representation. It is clear that a rich case base, encoding all major product types in a design domain would be the centerpiece of such an approach. EDAT (Electronic Design Assistance Tool) is intended to assist in precedent based design in the studio with the potential of expansion into the office setting. EDAT has been designed using object oriented system development methods. EDAT was used in a design studio at Carnegie Mellon University, during Spring 1996, and will be used in future studios, as well.

BACKGROUND

The introduction of computers into design especially in the world of practice came with the implicit promise that this would propel design to new levels of productivity. Yet, after nearly four decades of work, there has been little impact on design. Other than the notable exception of computerized drafting and visualization of designs, computers have almost completely bypassed their intended goals, such as, rapid design generation, elimination of mundane and repetitive tasks, and analysis based evaluation of designs, especially in the early stages. This picture also leaves design education-regardless of whether this is in the urban, architectural, civil or industrial design domain-in a state of 'lagging productivity', one not very different from where it has been all along. This perception is based on several well known symptoms: too high a level of effort in instruction, limits on transferability of design knowledge, long contact hours, and excessive dependence on design media. Computer assisted instruction, contrary to most expectations, has not helped this situation. Once again, presentation and visualization aspects of work has been positively influenced; but this has not translated into commensurate improvements in design quality.

Since, students do not have substantial design experience of their own, they learn by studying the successes and failures of other architects. In the past, educational systems were based on the master-apprentice relationships directly supporting precedent based learning. Presently, architectural education has evolved into a more didactic and abstract form. Masters of architecture who, in the past, served in the design studios of notable schools have been replaced by academics who do not and cannot sustain exceptional design practices. Thus, the student's learning in the design studio is based to a large extent on the understanding of important historical precedents or designs generated by those outside of academia. Consequently, a principal role of current design instruction is to expose students to a rich repertoire of outside precedents. We believe this can be done effectively in a computer medium.

A primary objective of the work described in this paper, then, is to devise new approaches that can undertake such a task. Our particular focus is the case based approach that can place into the electronic realm what designers do when they manually use design precedents. In many settings, design takes the form of adapting known solutions (whether these entail sizing of stair treads and risers, or composing with the verticals and horizontals of an elevation) to satisfy new problems. Part of learning to design in this mode then involves the learning of procedures for finding, adapting, creating, and saving design cases or precedents. A common form of instruction to accomplish this end is the explicit use of case dependent solutions in design. We call this case based instruction (CBI).

We postulate that some of these desired improvements can be brought about through CBI. It appears that we are not alone in holding this belief. Recently, systems that deal with case based reasoning in design have become popular. For brevity, we will cite three such systems to illustrate the range of possibilities. Maher's (1985) work is the first systematic exploration of case based technologies for building design. She developed a system called HI-RISE which assists in structural design using design schema based on prototypical solutions. Kolodner *et.al's* (1993)

work, on the other hand, uses building design evaluation cases as the basis for design decisions. While there is no accompanying computational design environment, their system called ARCHIE provides a well annotated catalog of successes and failures of notable designs. Finally, SEED (Flemming, *et.al.* 1995) a system being developed at Carnegie Mellon University, supports the early stages of building design through a case base of designs previously generated by the users of the system. Here the emphasis is on sophisticated matching, retrieval and adaptation functions that reside inside a comprehensive layout and three-dimensional design configuration environment. Perhaps the best matching precedent for our work in the field is Design-MUSE by Domeschek *et.al.*, (1994). This system assists designers through the conceptual stages by providing "an on-line library of design experiences." There are other case based systems that we could discuss here (such as, Oxman, *et.al.*, 1993, and Gero, 1993), however we choose not to due to space limitations and relevance to the specific goals of this work.

Each of these systems approach case based technologies in fundamentally different ways. HI-RISE uses precedents defined by the system developers to design new building structures. ARCHIE uses a rich case base, encoded also by system developers, but does not support a computer based design environment for case adaptation. SEED, on the other hand, both supports a case base-one normally developed by the users as opposed to system developers-and a computerized adaptation environment. Design-MUSE provides a versatile environment for creating and maintaining domain specific case based design aids. Obvious conclusions to draw from these are that there is growing interest in this approach and that the current work represented by these systems provides an important foundation for future work in this emerging field.

CASE BASED APPROACH TO EARLY DESIGN

Precedent Based Design

Precedents are essential starting points in design. A precedent provides for designers some guidance with which they can avoid the problem of 'reinventing the wheel' and reach satisfactory design solutions with relative ease. This implies several different strategies of matching a precedent to a problem at hand.

- *Solution Matching*: the prototype provides a solution which can with little modification be used to satisfy the current design problem.
- *Subsolution Matching*: the prototype provides subsolutions which can be synthesized, into a solution with scope greater than those of the individual subsolutions, in order to satisfy the current design problem.
- *Search Space Matching*: the prototype represents a domain of search which can be used to limit the scope of investigation relevant to the current design problem.
- *Process Matching*: the prototype illustrates an approach or method which can be applied to solving the current design problem.

Using one or more of these strategies, the designer can insure that some basic performance criteria are met within a reasonable amount of time. This approach is not without its critics. Some, for example, fear the potentially limiting effect of precedent based design on creative and inventive solutions. However, there is no concrete evidence that supports the validity of such a concern. If anything, there is ample evidence documenting that designers actively and frequently use precedent based processes (Akin, 1986; Mitchell, 1990; Heath, 1984).

Case Based Design

Before a case based approach to design can be effectively harnessed in the computer it must be formally represented. There are several difficulties with this. First, with most building types, there is a very large set of instances that illustrate them. Second, even if one were able to include a plethora of examples in a case base, not all of these would be relevant for a specific design problem. In a museum design project, for example, we may be interested in responding to issues, such as, use of modern materials, new construction techniques, advances in art preservation, and changing museum visitation patterns, most of which would be absent from many historical examples of museum design. Furthermore, other building types (of the non-museum variety) may have features that are relevant, such as buildings in a similar climate, buildings surrounding the site, buildings appealing to the owners, and so on. Thus selecting relevant cases constitutes a complicated matching operation. Therefore, documentation of cases in a computer has to be done to respond to a variety of user perspectives. Many building types, many instances

in each type, and many different attributes of these types and instances should be represented. The interconnections between cases and store/access operations that need to be carried out for each case must also be supported. Once a match is found there is the further problem of adapting the matching case to the problem at hand. Assuming that one can develop a CBI system that performs these tasks, there is the separate question of how to support learning in design through CBI. Our intent is to support and shed further light on design education through the exploration of the potential role of CBI systems and tools.

Cataloging Standards of Practice. One of the important uses of a CBI is to encode landmark designs in a given field. This can support the conventional form of precedent based design using exemplary designs. There is also great value in building case bases around the work of individuals and groups in an office setting. For example, if a firm is specialized in designing health care facilities, employees learn cumulatively about the do's and don'ts of health care design, and 'good' and 'bad' solutions to health care design problems. These design practices are more often than not reflected in a particular design solution rather than in the form of general principles. Saving and reusing an historical array of design cases in an office can be of immense value towards establishing internal standards of design practice. In this way, solutions developed at one time or by one individual in the office may be easily disseminated to others. This can work just as effectively in the studio setting. A case base used to save cases from a cross section of studios or in a given studio over time can become a powerful pedagogic instrument.

Design Presentation. Another important aspect of case based precedent documentation is its potential value in promoting presentation of designs in the computational medium. Computer Aided Architectural Design (CAAD) applications, having been introduced into the traditional design environment of drafting boards, parallel rules, and tack boards, appear to emulate these manual forms. Drafting systems emulate parallel bar operations and entering of data line by line. Drawing representations primarily use the paper-pencil metaphor. Presentation still relies heavily on hard copy output. While there are many practical reasons that justify these approaches, they prevent the new medium from developing into its own and reaching its ultimate potential. We envision that the basic CBI functionalities of saving and browsing cases is akin to asking for and obtaining information or explanations about a design. Thus, we see a potential use for a CBI which so far has not been exploited by any previous case based system: presenting of new design work to clients, reviewers and teachers. We believe a well documented case, stored within a case tool can support a range of useful design tasks (such as, generating, evaluating, reporting on design proposals) and lead to more productive interactions between design instructors and students than current manual or CAD based ones do. We expect that our work will support this functionality as well.

THE ELECTRONIC DESIGN ASSISTANCE TOOL (EDAT)

Goals of EDAT

The goals that initiated EDAT's development are:

- *A centralized store for research documentation gathered by students in the early stages of design.* This student documentation would tend to relate to design issues and building types being addressed in that year's studio.
- *A tool to perform performance analyses of this gathered design data, using third party applications.* This would require that students gather particular data in a pre-specified format. This aspect of EDAT's functionality has not yet been implemented.
- *A presentation tool for students' electronically produced design documents.* Here the browsing process that students engage in to find particular information about buildings in the case base, would be adapted to present work. Students would have the responsibility to organize the structure of their designs within the case base such that important aspects of their designs are legible and easily accessible using a browsing process.

Object orientation with OOSE

EDAT was designed using an object oriented methodology called OOSE (Object oriented software engineering) as developed by Jacobson, *et. al.* (1992). Although there are several competing object-oriented design methodologies, OOSE was chosen because of its emphasis on building user-centric systems. OOSE's general intent, as with all design methodologies, is the creation of robust, extensible and usable software systems.

In the OOSE method, the development of systems is a matter of designing five successive models of increasing levels of specificity to a particular software and hardware environment. These are called the requirements, analysis, design, implementation and testing models.

The *requirements model* specifies the overall system design objectives extensively. In the following sections we will discuss the development of this model for EDAT in particular detail. The *analysis model* abstracts all required data types and functionalities of the system, fully specifying the objects and the methods they require. The *design model* spells out the interactions that are needed between the objects in order to satisfy the requirements. All of this is carried out without a firm commitment towards a particular implementation environment. Next, the *implementation model* plans the actual coding of the system using a particular object-oriented (or in our case, some other) programming language. Finally the implemented system is *tested* to verify that it indeed delivers the required performance.

The Requirements Model

The overall description of the functionalities of a proposed software system in OOSE, is developed in the requirements model. This includes three components: the use case model, description of classes of users (actors), and object descriptions which also give an indication of how users could perform these use cases within an interface design. In specifying the requirements model in EDAT, we defined three key concepts that constitute the core of user interactions with the system: facts, topics and topic trees.

Facts are units of information that constitute the lowest common denominator in EDAT's case base. They contain information about graphic as well as performance aspects of a design (plans, sections, photos, and acoustics, heating, cooling, respectively); and come in four forms: raster, CAD drawing, short text and long text.

Topics are variables that describe the category in which a fact belongs. Topics are implemented as delimited strings which represent 'leaves' within a tree structure. Each building type would have one such collection of topics forming its own virtual 'topic-tree'.

Topic trees are structures within the EDAT database arranging the topics into a hierarchical tree. This hierarchy is described as a series of topic trees with building types at the roots, building names (i.e., Kimbell Art Center, etc.) as subcategories of building types and topics/subtopics at the lowest levels of the tree.

Use cases of EDAT. OOSE methodology assumes the primacy of a construct called *use cases* which assist in specifying the desired functionality e software system. Use cases model specific interactions which users of the system may wish to perform with the completed system. Use cases are not dependent on peculiarities of a specific implementation environment and are created from a user's point of view. These two factors help make the requirements model abstract such that it can be implemented in virtually any programming environment and user-centered such that user's needs take primacy over other considerations. Use cases constitute a high level, user-centered specification which forms a thread that connects all the stages of the OOSE method.

EDAT's use cases were organized under four categories: *system level, browse, build case* and *build case base*. The system level use cases handle functionalities such as entering and exiting EDAT, printing and help. The central use cases in the present implementation of EDAT involve the browse use cases. Browse involves two procedures: first defining a browse filter set and then submitting this set through EDAT's query processor which develops a SQL query. This composed query is then submitted to case base and all facts which satisfy its constraints are returned as a listing in the Browse and Search Results window in the main EDAT interface. The Browse and Search Results window lists a description of the fact and the type of fact - whether text, cad, raster or composite. (Figure 1).

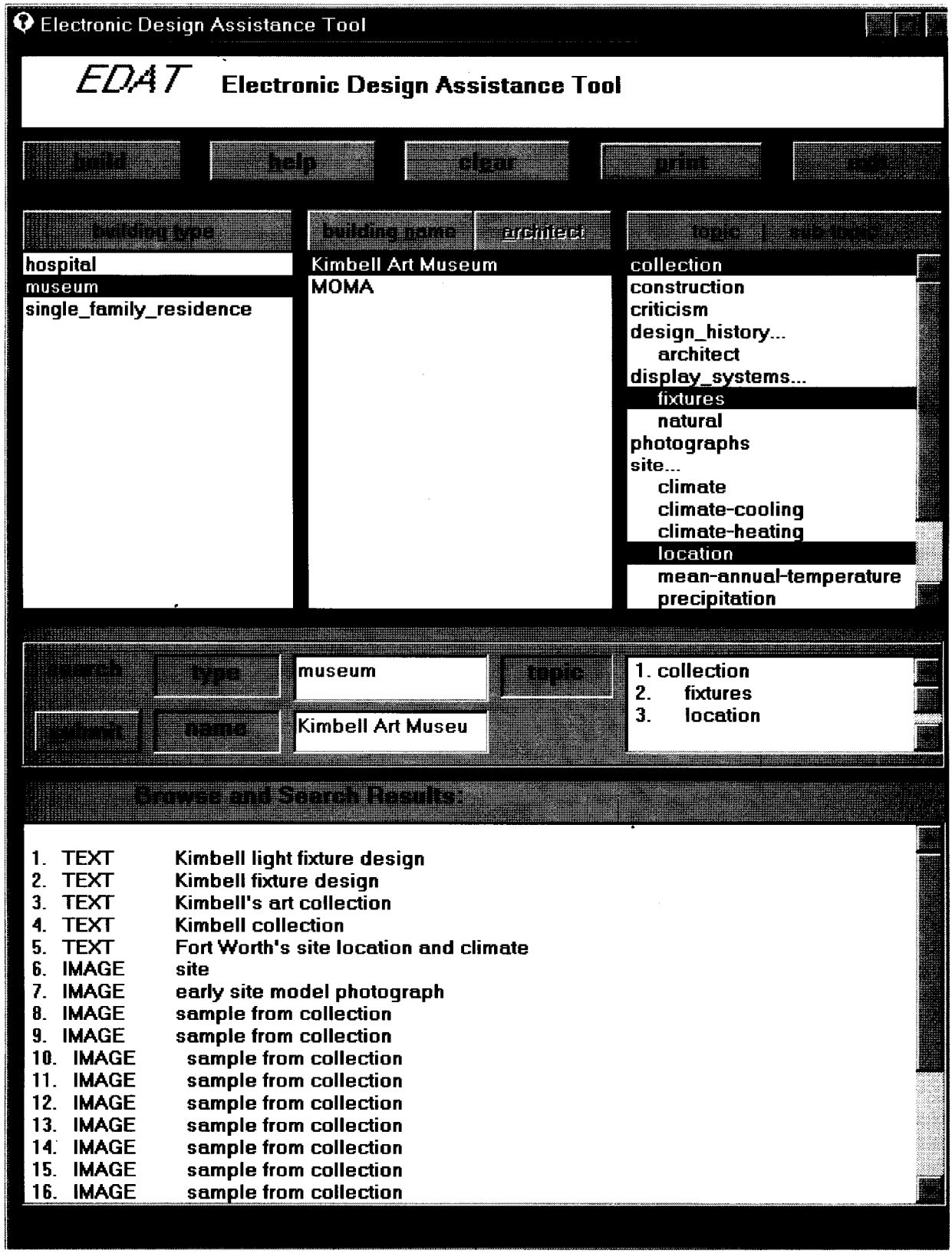

Figure 1: The primary interface window of EDAT for case browsing, implemented in Visual Basic

Actors. Actors are the roles a user might play in using the system. Within EDAT we have identified three roles: *student user, case builder*, and *case base manager*. Any single person could play all three of these roles at different times. The student user, a student in an architectural design studio, would be the main user of EDAT. Typically the student user would research building design and store their own designed work within EDAT. The case builder is the person who actually adds new data to the EDAT case base, while the case base manager is the person responsible for maintaining data integrity: assuring that inappropriate data is not added or that valued data is not deleted.

Interface design. The interface of EDAT was designed to follow standard Microsoft Windows GUI conventions. Most interaction is through the selection of buttons, highlighting items in scrollable lists and the viewing of retrieved information in resizable windows. The interface is further described in the following sections.

The Object Model

In OOSE nomenclature these there are three kinds of objects: *entity, interface*, and *control.* The entity object model which will be described here corresponds to all objects which store persistent data in the case base. (Figure 2) In EDAT there are six basic data entity objects: facts, building cases, architects, building types, topic nodes and data sources. Facts comprise four different types: text, cad, raster and composite. The composite type is a container for an aggregation of any of the four fact types, including other composite facts. This technique of having a subclass container holding other elements at the same level of the hierarchy including itself, is described as recursive composition (Gamma, 1995).

Implementation

EDAT was implemented within a Microsoft Visual Basic 4.0 (16-bit version) program shell. EDAT's database was created with Microsoft Access 2.0, a relational database. Implementation using the 16-bit version of Visual Basic 4.0 was required to allow the installation of EDAT within the Windows 3.1 environment currently implemented at the computing facilities of the Department of Architecture. We welcomed this platform as it is widely available and easy to access by many potential users of EDAT.

The system architecture of EDAT consists of four main components: 1. a user interface containing both a primary interface (used to browse the database) and a secondary interface created from within Access (used to modify the database) 2. a system control component handling event-driven, user-interface interaction 3. a query processor linking the user interface and the database in the browse process 4. the database.

Entity Objects. The query processor in EDAT retrieves data from the database without the hard-coding of queries in the interface. Changes may be easily made to either the database or the interface independently of the other. Changes to the query processor may be made as required to maintain communication between all components of the system.

In EDAT data is stored in a relational database. Therefore, as a step in the creation of the database, we mapped the object model onto appropriate relational tables. There are several ways of doing this mapping (Rumbaugh, 1991) but the way chosen in EDAT, is perhaps the simplest, that is, each object was given its own relational table while associations between objects were implemented as relationship tables in the manner of standard entity-relationship database modeling (Date, 1995). Hierarchical relations are implemented as cascaded ID attributes which are automatically entered into subclass tables.

An important part of the case-building process within EDAT is the design of a topic tree for each building type. Different building types may share various topics, but the assignment of each topic to each building type is at present a manual process in EDAT. There is no prescribed structure inherent in the EDAT design for these topics and the virtual topic-trees which contain them in their nodes. This is one of the greatest strength of EDAT: its basic identifiers or indexing of subject matter is completely user defined. It does not assume that information must be organized in any particular way. However this places on the case-builder the burden of creating a coherent topic tree. Since the manner in which facts are indexed has a great effect on how these same facts are retrieved from the case base, the conceptual process of designing a topic tree for each building type, lies at the heart of both information retrieval and information storage in EDAT.

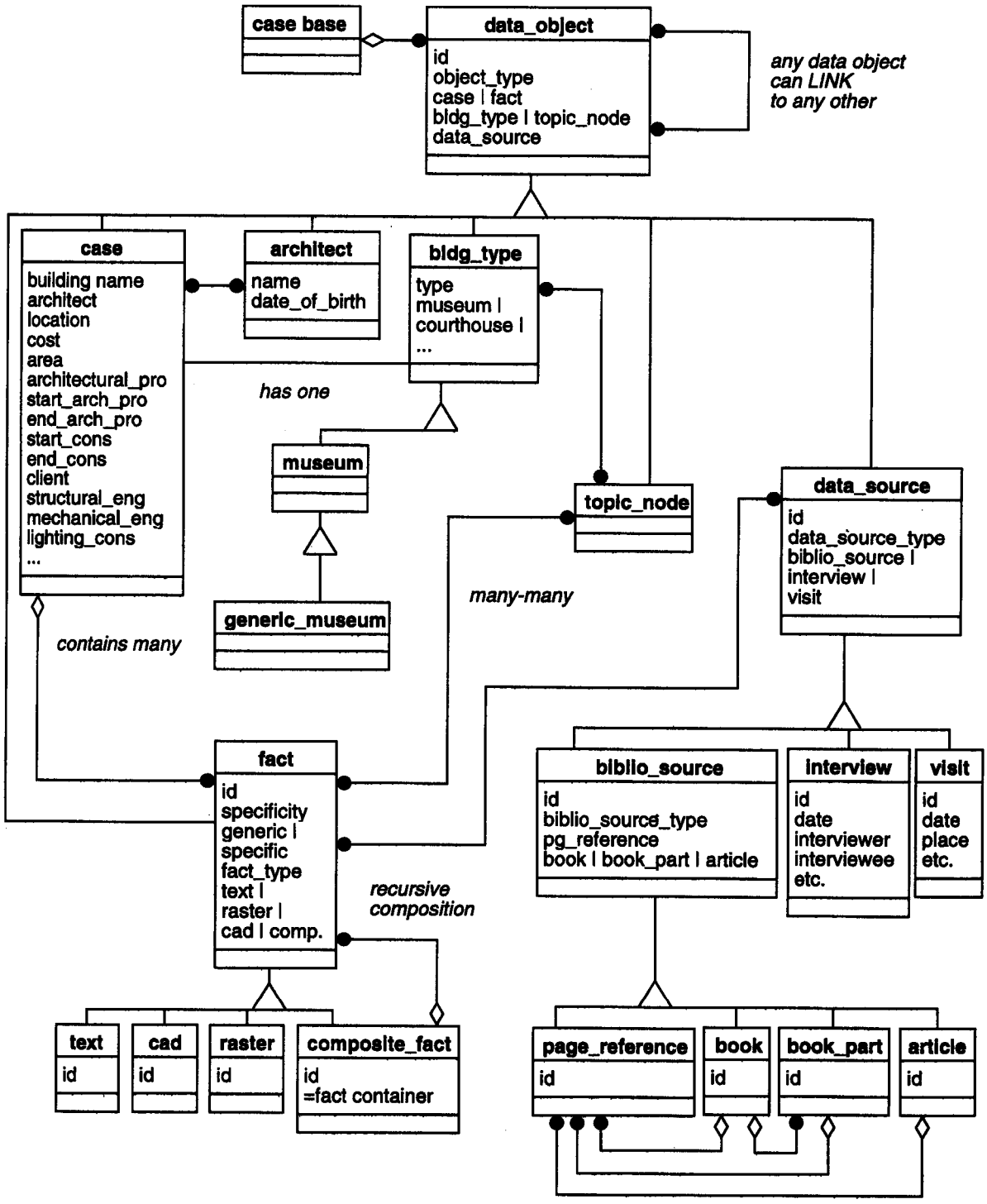

Figure 2: EDAT object model shown in OML notation (from Rumbaugh, *et.al.*, 1991).

Interface Objects. The primary interaction that the student users have with the EDAT system is through the main EDAT case-browsing window. (Figure 1), This window primarily provides for the processing of queries to the database in the form of browsing. The main EDAT case-browsing window consists of: 1. a menu bar with 'Help', 'Clear', 'Print', 'Exit' and 'Build' (to connect to the database) buttons. 2. browse windows with building type, building name, architect, and building topics/subtopic filters 3. a submit panel showing all user-selected fields from the browse windows 4. a submit button used by the user to send all selected fields as displayed in the search windows to the query processor for processing 4. a browse and search results window displaying the facts retrieved from the query processor in the form of a file by listing of facts retrieved by the query processor.

Browsing the database in EDAT is accomplished in the browse windows located in the top half of the interface. Specific browse filter criteria may be specified for each of these windows. These filter criteria are: building type, building name, architect, and topic. The criteria can be applied in any order and any or all can be omitted. As one clicks on the buttons of Building Type, Building Name, Architect or Topic/Subtopic, the case base is queried for these items. For example when one clicks on the Building Name, all building names are returned which also satisfy all previous selections. If no other items had been selected then all building names in the case base will be displayed. Browsing is therefore an additive process of constructing more and more restrictive constraint sets. The more restrictive the filter set becomes, the fewer the number of facts returned from the case base.

Selection of the 'Build' button on the menu bar brings up a working Access 2.0 interface with the EDAT database loaded as the current project. This interface provides access to a series of screens (Figure 3) allowing the user to add to or modify the database. Modification of the database refers to the addition or deletion of topic nodes from the database structure. To add data to existing topic nodes in the database, the user must identify the data type (raster image, text file, AutoCad file, etc.) and provide a full path to the current location of the file to allow later retrieval of the file from the database. The screens for the addition of data to the database allow the user to input information into both base and relationship tables from a single form.

Fact windows are launched by the user's selection of specific facts from the listing in the browse and search results window. Each database fact is launched in a separate window for viewing. Fact windows may contain short text facts, text files, raster images or AutoCad drawings. Raster images are displayed as bitmap images. AutoCad files are treated as embedded objects using OLE (Object Linking and Embedding). The selection of an AutoCad file from the search results window launches a fully working AutoCad user interface with a copy of the selected AutoCad file as the currently active file. The user may manipulate this file as desired within the AutoCad window and save the file to another directory without altering the original drawing in the database.

DEPLOYMENT

EDAT is currently deployed in an undergraduate computer studio within the Department of Architecture. The studio contains ten PC's, each assigned to an individual student. Each PC has a 120 MHz Pentium processor, 24 Mb of RAM, a 500Mb hard drive, and Windows 3.1 as the operating system. The machines are networked to a common server. Existing EDAT data (at this time limited to the Kimbell Art Museum case) resides on the server. Each student machine is provided with a copy of the EDAT program (launchable by selection of an icon in the standard Windows manner). Currently, all additional data entered by students resides on the individual PC's rather than the server. As EDAT is further refined, all data storage will be on the server, allowing students access to material entered into the system by other students.

LESSONS LEARNED

One advantage of object-oriented design in general and OOSE in particular is that modularity is inherent to the design. This generally means that not all use cases need be implemented to create a functional system. One can choose to implement a subset of the designed use cases without the logic of the entire system failing. This proved to be the case in the implementation of EDAT as we added features (such as, a button labeled "architect" lists building cases for each architect in the database) and subtracted features (a graphic representation of the topic tree, or s text-based search window) from it.

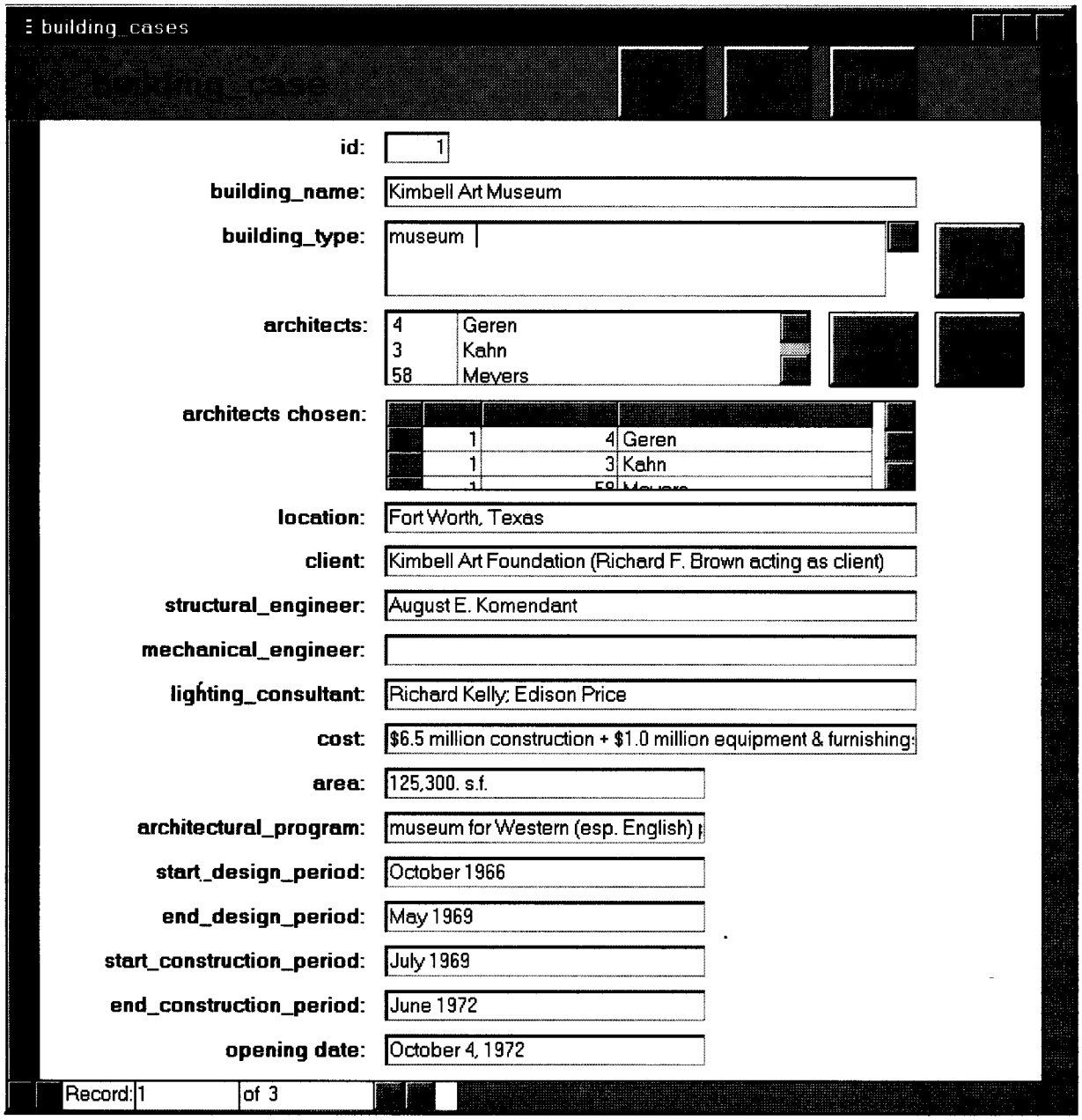

Figure 3: The primary interface window of EDAT for case building, implemented in ACCESS

All five OOSE models were completed in succession during the design of EDAT and were found to be quite useful and appropriate for each stage of EDAT's development. The formality and rigor of such a systematic approach enabled us to implement the entire system interface in a matter of three weeks which is, to a large extent, due to the time we spent designing it (three times the implementation time).

In due time, we expect to learn more about the use and usability issues in EDAT. As students continue to use EDAT in case analysis and design presentation work, we expect to learn about the adequacy of the case building and browsing functionalities that are provided. Similarly, the interface design we provided, no doubt, requires improvements. We expect to learn about the successes and failures of EDAT in both respects from the users in the design studio, over a period of one year. A formal evaluation program is being developed, including interviews and questionnaires in addition to the day to day contact with students. Furthermore we will use EDAT to accumulate the

design work over a period of time in the form of student designed cases. We expect this to be an important aspect of EDAT's pedagogic experiment.

Several suggestions have already been received about expanding the functionalities of EDAT and deploying it more broadly. One of the functional expansions suggested is the inclusion of a HyperCard like browsing function that would eliminate going back to the browser and using the filters each time a new target fact is selected. Another one is the accommodation of facts in multi-media format, including animations, video and audio recordings. Both of these extensions are possible within the present architecture of EDAT and seen as potential directions of extension. Another suggestion made by present users of EDAT is that it should be launched on the World Wide Web. This would attract a wider audience of browsers and case builders. Undoubtedly, this would potentially increase the value of EDAT as a design tool and enrich its case base considerably. However, we do not expect to undertake this step before it is more completely tested and copyrighting issues surrounding the material included in the case base are resolved.

ACKNOWLEDGMENTS

We wish to thank the following people for contributing to EDAT: Carnegie Mellon University, Provost for Education's Courseware Project which provided funding for EDAT; James Garrett Jr. and Steven Fenves for effectively guiding the course work supporting the development of EDAT through the OOSE process; Zeyno Aygen, Eugene Kim and Emre Ilal for their contributions to the design and implementation of EDAT; the advisory committee of the EDAT project: Ardeshir Mahdavi, John Decker, Daniel Rehak and Rudi Stouffs for their valuable comments.

REFERENCES

Akin, Ö. (1986). *Psychology of Architectural Design*, London: Pion Ltd.

Date C. J. (1995). *An Introduction to Database Systems, Sixth Edition*. Reading, MA: Addison-Wesley.

Domeshek, E. A., J. L. Kolodner, and C. M. Zimring (1994) "The design of a tool kit for case-based design aids" in *Artificial Intelligence in Design* '94 ed. by J. Gero, Kluwer Academic Publishers: The Netherlands.

Flemming, U., Aygen, Z., Coyne, R., Snyder, J. (1995). "Case-based design in a software environment that supports the early phases in building design" Unpublished Report, Department of Architecture and the Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

Gamma, E., Helm R., Johnson R., Vlissides J. (1995*). Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley.

Heath, T., (1984). *Method in Architecture*. New York: John Wiley and Sons.

Kolodner, J., (1993). *Case-Based Reasoning*. San Mateo: Morgan Kaufmann.

Jacobson. I. (1992). *Object-Oriented Software Engineering: a use-case driven approach*. Reading, MA: Addison-Wesley.

Maher, M. L., and Fenves, S. J., (1985). HI-RISE: an expert system for the preliminary structural design of high rise buildings, in J. S. Gero (ed.*). Knowledge Engineering in Computer-Aided Design*. Amsterdam: North Holland.

Mitchell, W. J., (1990). *The Logic of Architecture*. Cambridge, MA: The MIT Press.

Oxman, R. E. and R. M. Oxman (1991) "Refinement and adaptation: two paradigms of form generation in CAAD" in *CAAD Futures '91* ed. by Gerhard Schmitt, Friedr. Vieweg & Sons, Wiesbaden.

Rosenman, M. A., J. S. Gero, and R. E. Oxman (1991) "What's in a case: the use of case bases knowledge bases, and data bases in design" in *CAAD Futures '91* ed. by Gerhard Schmitt, Friedr. Vieweg & Sons, Wiesbaden.

Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorenson W. (1991). *Object-Oriented Modelling and Design*. Englewood Cliffs, NJ: Prentice Hall.